



# **CS 161**

## **Intro to CS I**

### Recursion



# Odds and Ends

- Assignment 3 demo this week
- Peer reviews

# Exercise Pointers vs. References



Oregon State University  
College of Engineering

- What if you made a pointer (p2) that points to a pointer (p) to an int (x)?
  - What would the picture look like?
  - Write the code for this picture.
- Can you make this same picture for references?
  - What if you had two references, r and r2?



# Recursion

- What is it?
  - Function that calls itself 1 or more times (directly or indirectly)
  - Has 1 or more base case for stopping
  - Inductive reasoning: general case must eventually be reduced to a base case



# Example: Drawing Rectangles

- Iterative Solution:

```
void draw_rect(int i) {  
    for( ; i > 0; i--){  
        cout << "*****" << endl;  
        cout << "*          *" << endl;  
        cout << "*****" << endl << endl;  
    }  
}
```



# Example: Drawing Rectangles

- Recursive Solution

```
void draw_rect(int i) {  
    if(i>0){ //Base case  
        draw_rect(--i); //Recursive call  
        cout << "*****" << endl;  
        cout << "          *" << endl;  
        cout << "*****" << endl << endl;  
    }  
}
```

# What is different when we call after?



- Recursive Solution

```
void draw_rect(int i) {  
    if(i>0){        //Base case  
        cout << "*****" << endl;  
        cout << " *           *" << endl;  
        cout << "*****" << endl << endl;  
        draw_rect(--i);    //Recursive call  
    }  
}
```



# Example: Factorial

- Definition

$$0! = 1;$$

$$n! = n * (n-1) * \dots * (n-(n-1)) * 1 = n * (n-1)! ; n > 0$$





# Iterative Factorial

factorial(0) = 1;

factorial(n) =  $n * n-1 * n-2 * \dots * n-(n-1) * 1$ ;

```
long factorial(int n) {  
    long fact;  
    if(n==0)  
        fact=1;  
    else  
        for(fact=n; n > 1; n--)  
            fact=fact*(n-1);  
    return fact;  
}
```



# Recursive Factorial

factorial(0) = 1;

factorial(n) = n\*factorial(n-1);

```
long factorial(int n) {  
    if (n == 0)    // Base case  
        return 1;  
    else  
        return n * factorial(n - 1);    // Recursive call  
}
```

# Computing Factorial Iteratively



Oregon State University  
College of Engineering

factorial(4)

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Computing Factorial Iteratively



Oregon State University  
College of Engineering

$$\text{factorial}(4) = 4 * 3$$

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Computing Factorial Iteratively



Oregon State University  
College of Engineering

$$\begin{aligned}\text{factorial}(4) &= 4 * 3 \\ &= 12 * 2\end{aligned}$$

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Computing Factorial Iteratively



Oregon State University  
College of Engineering

$$\begin{aligned}\text{factorial}(4) &= 4 * 3 \\ &= 12 * 2 \\ &= 24 * 1\end{aligned}$$

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Computing Factorial Iteratively



Oregon State University  
College of Engineering

$$\begin{aligned}\text{factorial}(4) &= 4 * 3 \\ &= 12 * 2 \\ &= 24 * 1 \\ &= 24\end{aligned}$$

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Computing Factorial Recursively



Oregon State University  
College of Engineering

factorial(4)

```
factorial(0) = 1;
```

```
factorial(n) = n*factorial(n-1);
```



# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\text{factorial}(4) = 4 * \text{factorial}(3)$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n * factorial(n-1);
```

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= \underline{4} * (3 * \text{factorial}(2)) \end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * ( 3 * \text{factorial}(2) ) \\ &= 4 * ( 3 * ( 2 * \text{factorial}(1) ) )\end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * ( 3 * \text{factorial}(2) ) \\ &= 4 * ( 3 * ( 2 * \text{factorial}(1) ) ) \\ &= 4 * ( 3 * ( 2 * ( 1 * \text{factorial}(0) ) ) )\end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n * factorial(n-1);
```

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \end{aligned}$$

# Computing Factorial Recursively



Oregon State University  
College of Engineering

```
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);
```

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \end{aligned}$$



# Computing Factorial Recursively



Oregon State University  
College of Engineering

$$\begin{aligned} \text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \\ &= 24 \end{aligned}$$

`factorial(0) = 1;`

`factorial(n) = n*factorial(n-1);`

# Differences

- Pros
  - Readability
- Cons
  - Efficiency
  - Memory



# Recursive Factorial



factorial(4)

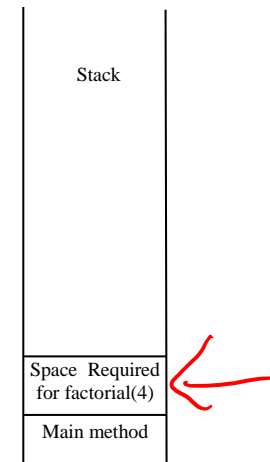
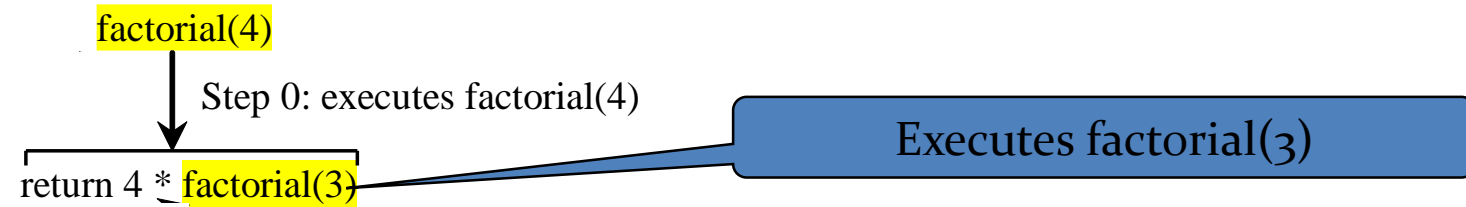
Executes factorial(4)

Stack

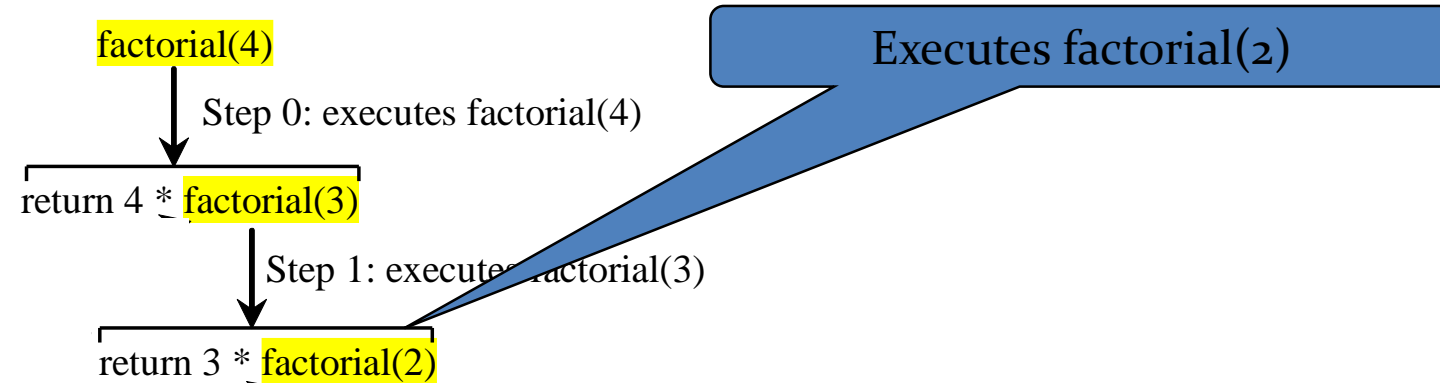
Main method



# Recursive Factorial

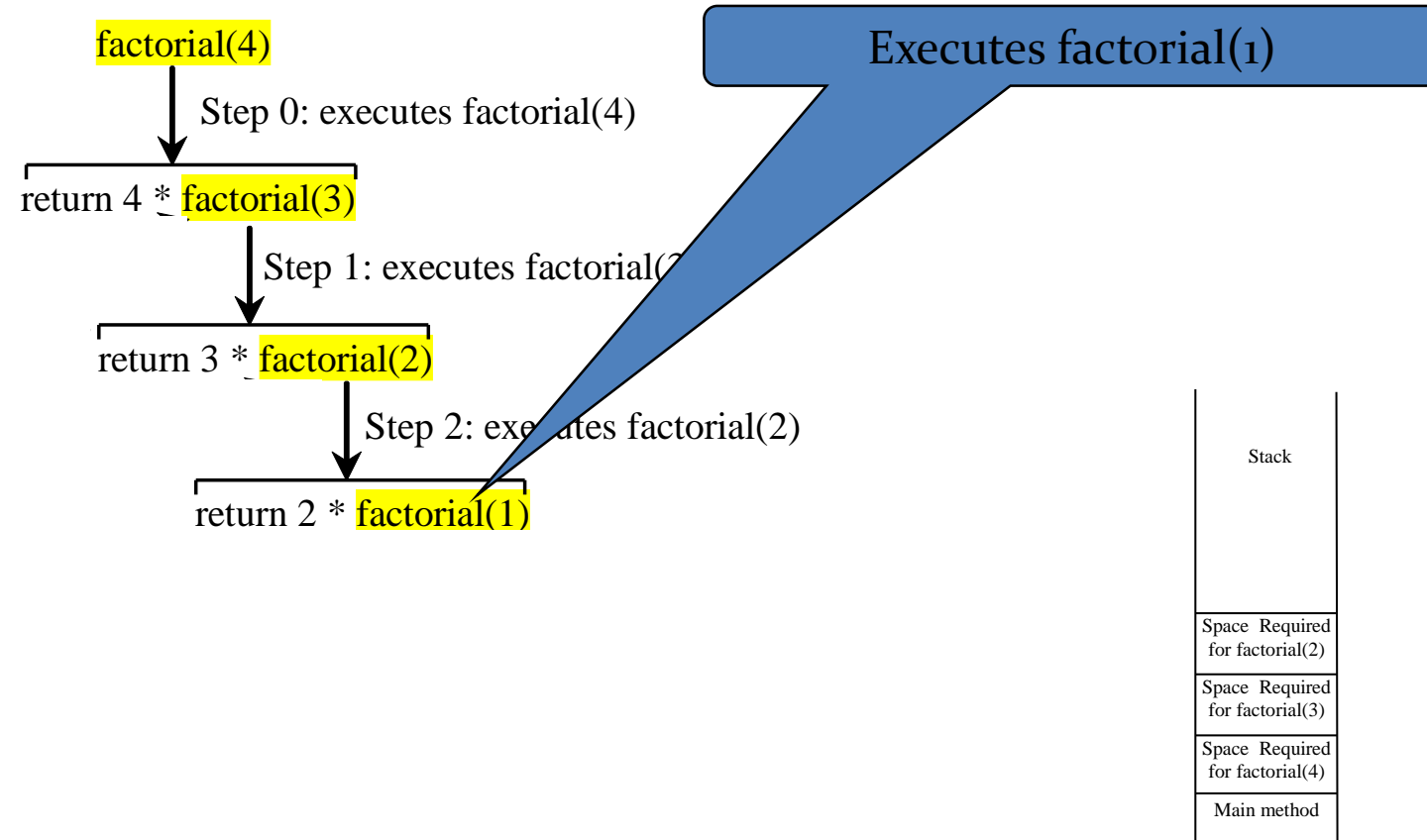


# Recursive Factorial

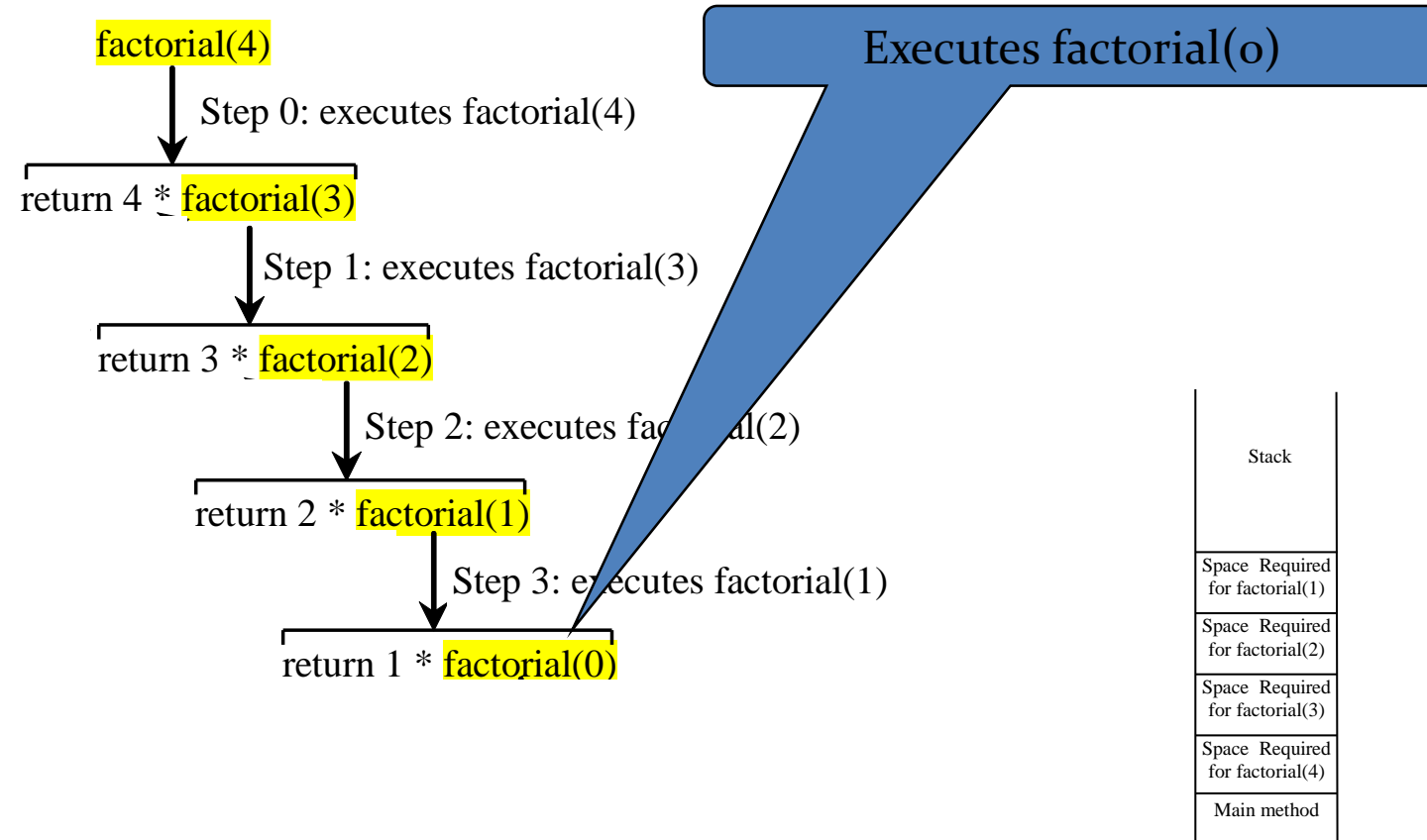


Stack
Space Required for factorial(3)
Space Required for factorial(4)
Main method

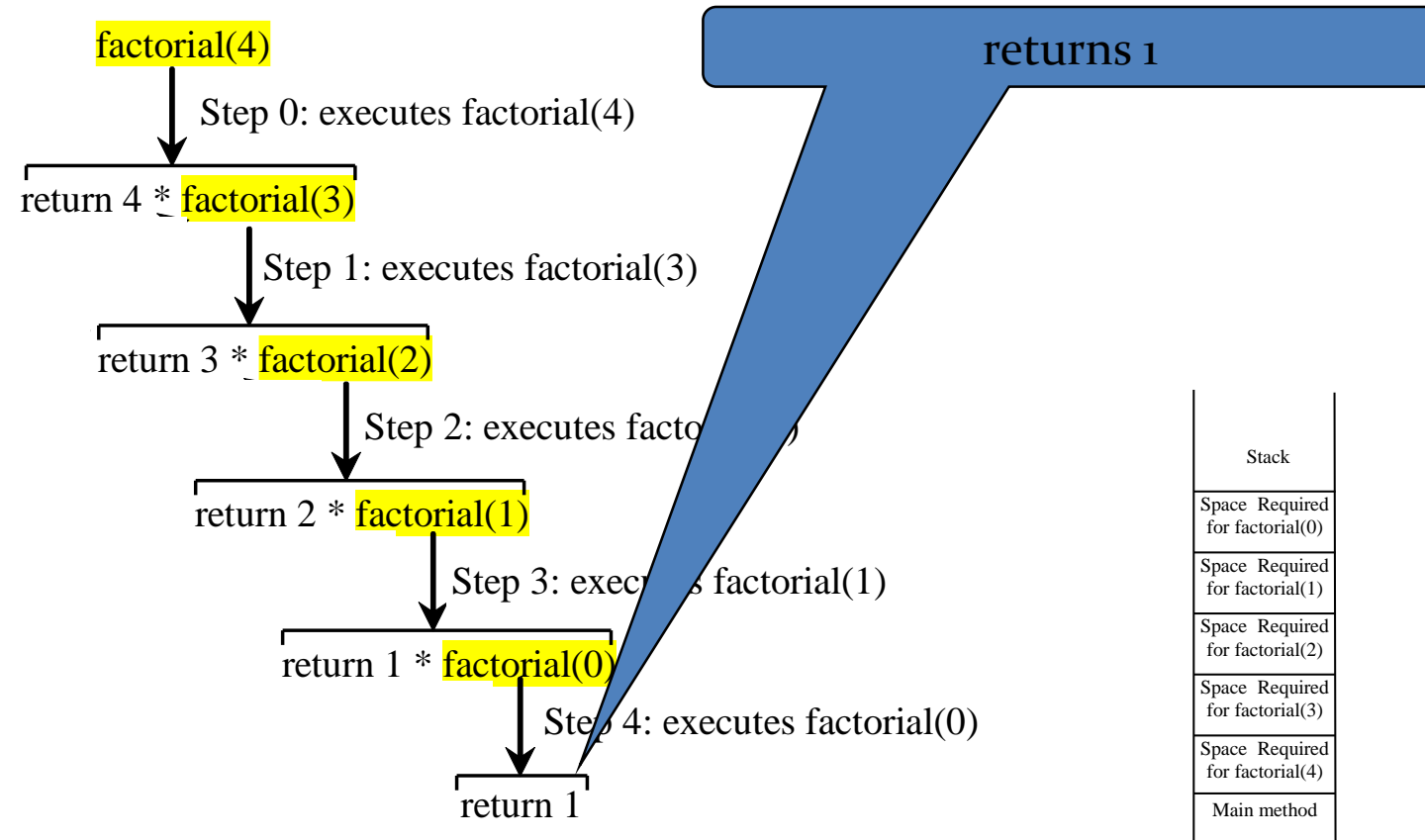
# Recursive Factorial



# Recursive Factorial

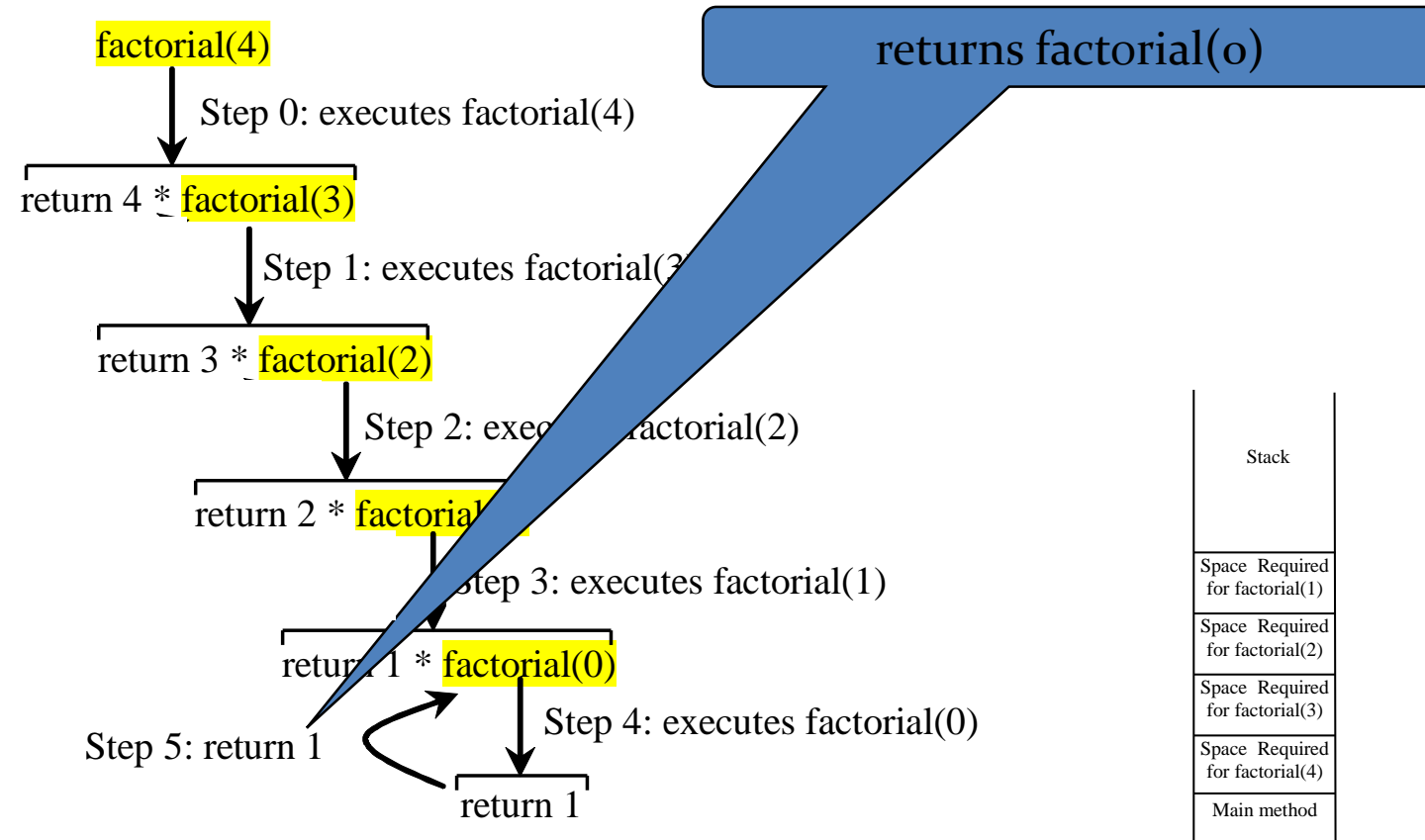


# Recursive Factorial



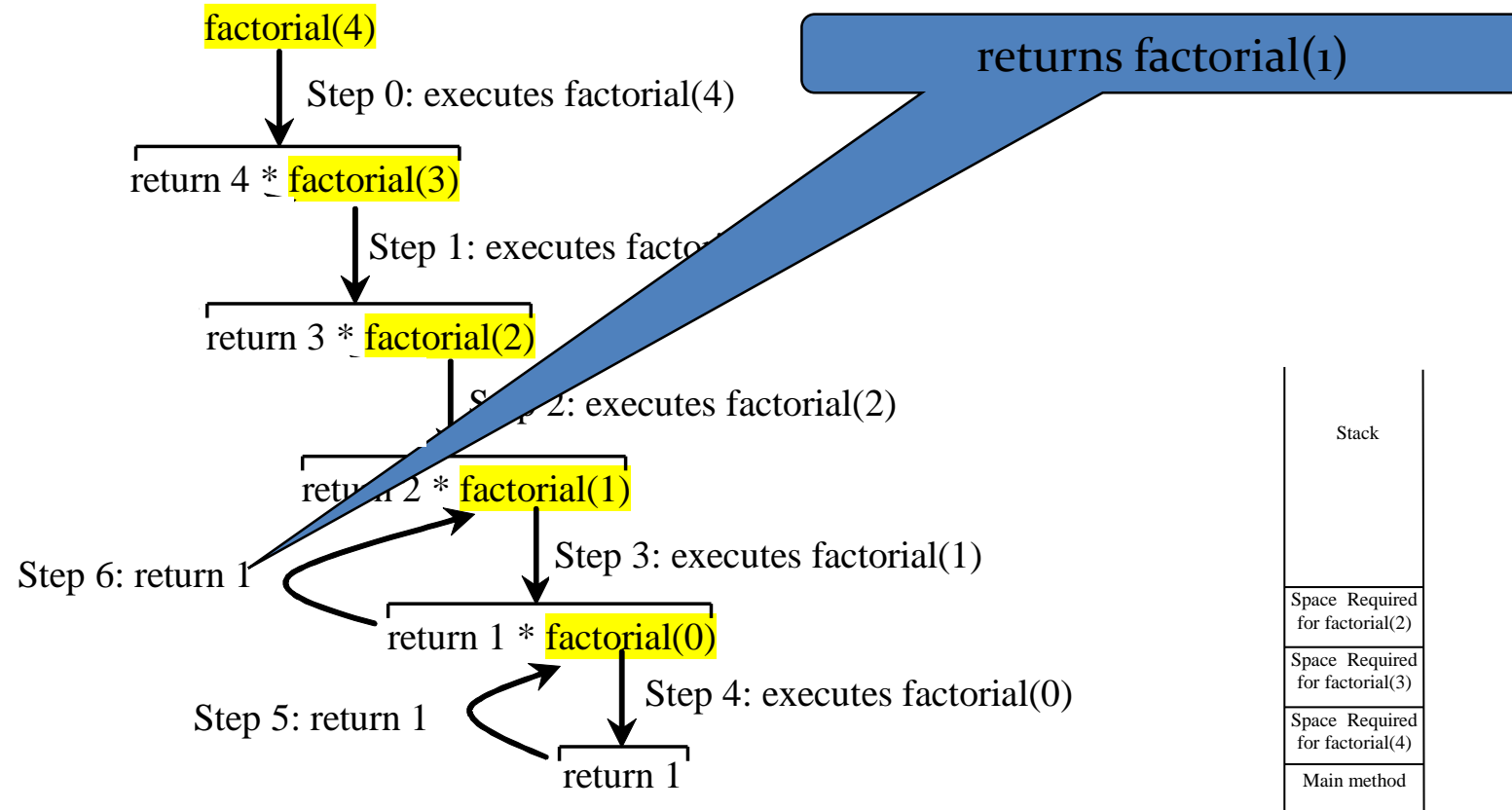


# Recursive Factorial

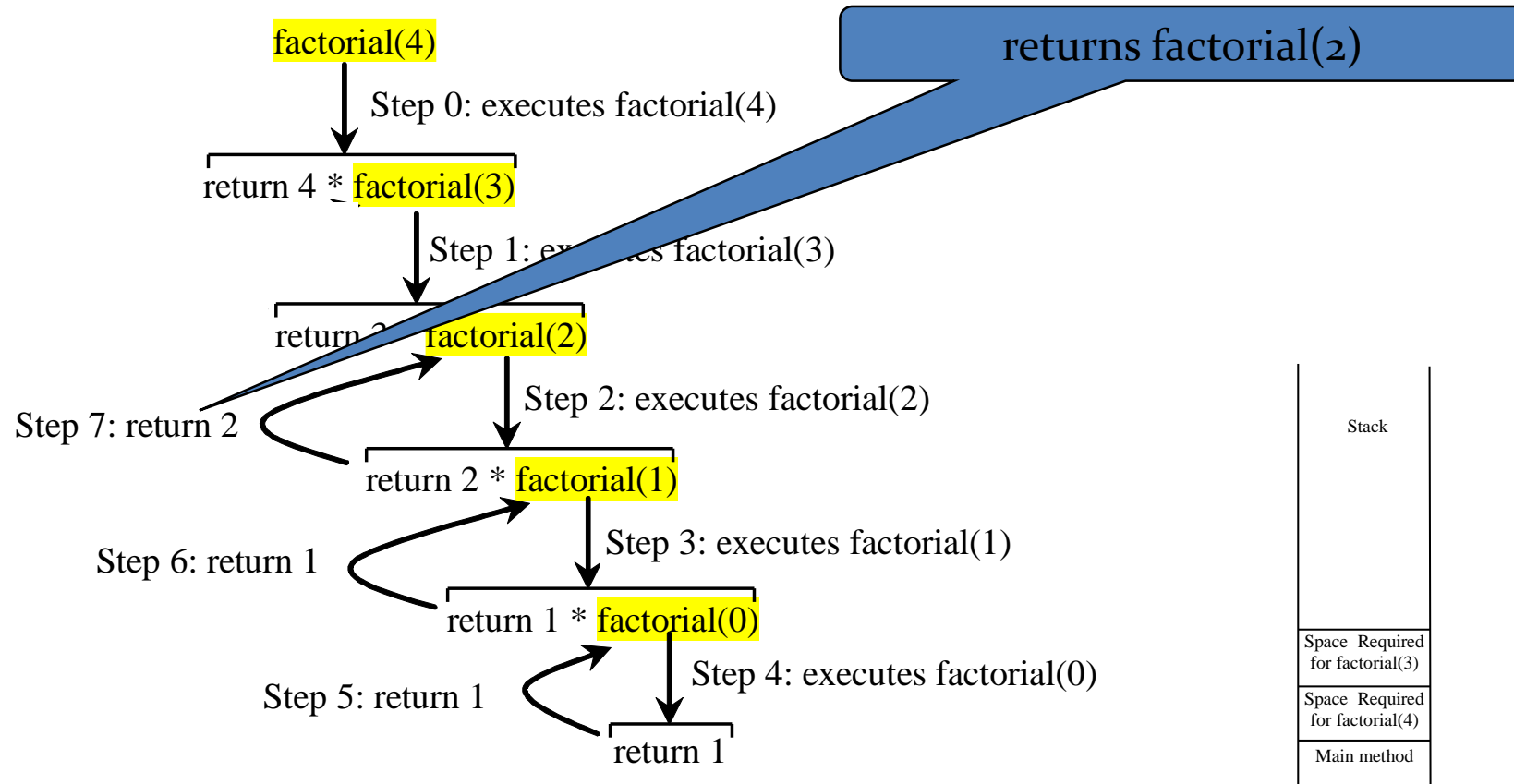


Stack
Space Required for factorial(1)
Space Required for factorial(2)
Space Required for factorial(3)
Space Required for factorial(4)
Main method

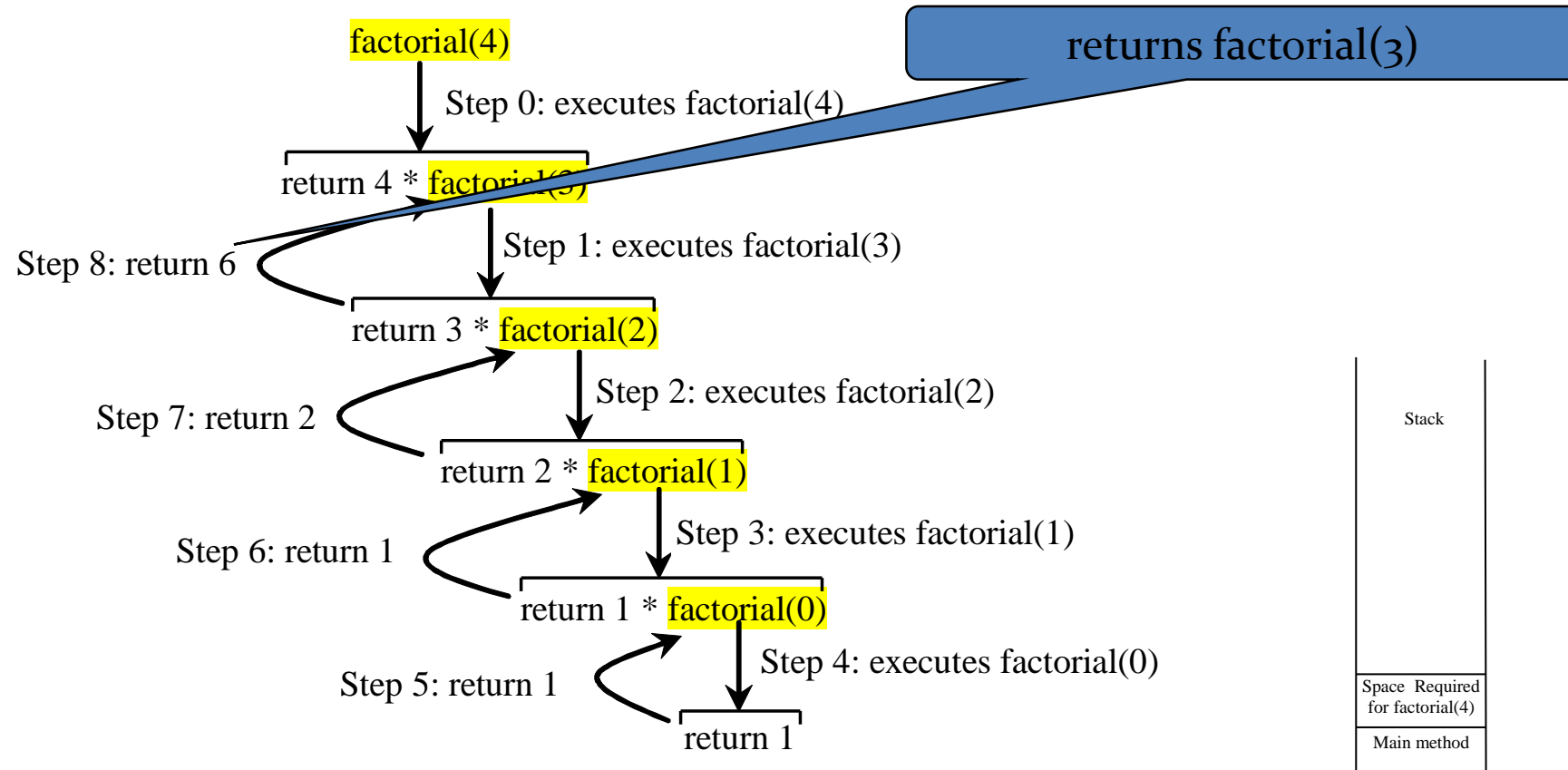
# Recursive Factorial



# Recursive Factorial



# Recursive Factorial



# Recursive Factorial

